VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AUTOMATIZACE ANALÝZY VÝKONU A SPOTŘEBY ZVOLENÉHO SYSTÉMU

AUTOMATIZATION OF ANALYSIS OF PERFORMANCE AND POWER CONSUMPTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ RUDOLF**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VOJTĚCH NIKL**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Rudolf Tomáš**
Obor: Informační technologie
Téma: **Automatizace analýzy výkonu a spotřeby zvoleného systému**
Automatization of Analysis of Performance and Power Consumption
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s moderními procesorovými architekturami a jejich možnostmi taktování procesoru, meření výkonu a spotřeby jednotlivých částí systému.
2. Navrhněte univerzální sadu skriptů, která na zvolených algoritmech automatizovaně změří daný systém z pohledu spotřeby a výkonu a naměřené výsledky přehledně zobrazí na výstup v podobě grafu, diagramu, tabulky apod.
3. Řešení implementujte.
4. Otestujte řešení na sadě dostupných benchmarků a systémů.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Nikl Vojtěch, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Tato práce se zabývá zvýšením efektivity superpočítačů. Vyšší efektivity lze dosáhnout pomocí snížení frekvence procesoru, pokud to daný algoritmus výrazně nezpomalí. Tato práce představuje sadu skriptů určených ke sledování spotřeby procesoru společně se skripty pro vizualizaci těchto naměřených hodnot. Dále také umožňuje jednoduché ovládání frekvence procesoru. Vytvořené řešení poskytuje uživateli možnost změřit efektivitu a optimalizovat výpočetní výkon počítače specificky pro jeho algoritmus. Díky této práci bude uživatel informován o tom, zda je výhodné provozovat jeho algoritmus na té či oné frekvenci procesoru.

Abstract

This thesis deals with increasing efficiency of supercomputers. Higher efficiency can be achieved by reducing frequency of processor if the algorithm does not slow down significantly. This thesis presents set of scripts designed to monitor consumption of processor along with scripts that visualize these measured values. It also allows easy control of processor frequency. The created solution gives user a capability to measure given algorithm efficiency and optimize computing power of specific computer exactly for the algorithm. Due to this work the user will be informed about whether it is advantageous to run his algorithm on one or other frequency of the processor.

Klíčová slova

Analýza, spotřeba, procesor, frekvence, paměť, PAPI, MSR, RAPL, Intel, Linpack

Keywords

Analysis, consumption, processor, frequency, memory, PAPI, MSR, RAPL, Intel, Linpack

Citace

RUDOLF, Tomáš. *Automatizace analýzy výkonu a spotřeby zvoleného systému.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vojtěch Nikl

Automatizace analýzy výkonu a spotřeby zvoleného systému

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vojtěcha Nikla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Rudolf
15. května 2018

Poděkování

Děkuji panu Ing. Voječhu Niklovi za neocenitelné rady v odvětví informačních technologií, které mi bylo před touto prací téměř neznámé.

Obsah

Kapitola 1

Úvod

Každý den běžného moderního života kohokoli z nás vyžaduje enormní výpočetní výkon na serverech a superpočítačích. Tyto počítače mohou simulovat vývoj počasí [27] ve vybrané oblasti, předpovídat tržní cenu akcií [14] na burze nebo jen sestavují playlist nejlepší hudby pro každého z nás. Všechny tyto činnosti stojí výpočetní výkon, jehož důsledkem je převod elektrické energie na energii tepelnou a tato tepelná energie musí být odvedena pryč od počítačů, což znamená ještě více spotřebované energie.

Stavbou ekologickych datacenter [1] lze docílit vyšší efektivity a zároveň využít odpadní teplo. Odpadní teplo z datacentra může být například využito k ohřevu domácností v dané lokaci [32]. Další možností jsou například úspory energie, která nemusí být převáděna ze střídavého proudu na stejnosměrný a nazpět v případě výpadku dodávky elektřiny, ale zabudováním baterií přímo na základní desky [3] eliminovat tyto zbytečné převody.

Existují však možnosti, jak zmenšit množství vyprodukovaného tepla efektivním využitím procesorů v těchto serverech. Takovouto možností je například snížení frekvence procesoru, kdy se se snížením frekvence může zároveň snížit napětí přiváděné k tomuto procesoru a zvýšit efektivitu ještě více. Toto řešení ale zákonitě způsobí to, že výsledek se musí počítat déle. Ve výsledku ale pro některé algoritmy toto řešení způsobí jejich vyšší energetickou efektivitu, protože jsou počítány na procesorech, které mají menší spotřebu díky snížené frekvenci a napájecímu napětí, a dojde pouze k malému spoždění než bude výsledek hotový. Teoretickým příkladem může být například procesor na vysoké frekvenci s vyšším napájecím napětím, který vypočítá výsledek o 10% rychleji, ale spotřebuje o 50% více energie, než procesor používající nižší frekvenci a napájecí napětí.

Dnes již nástroje zjišťující spotřebovanou energii a další parametry efektivnosti existují, ale jejich použití je složité a těžkopádné. Tato práce vnikla jako řešení tohoto problému a jejím cílem je vytvořit sadu skriptů, které využijí již existující nástroje a postupy a jejich výsledky interpretuje uživateli srozumitelným způsobem, například pomocí grafů nebo tabulek. Tato sada skriptů musí mít možnost být automatizována například pro vyhledání frekvence s celkovou nejnižší spotřebou a tuto činnost musí být schopná provést opakovaně s měnícími se parametry sledovaného programu.

Cílem této práce je vytvoření univerzálního terminálového programu, který se bude skládat z několika skriptových modulů. Součástí této práce bude předvedení naimplementovaných funkcí na známých benchmarcích a vyvození doporučení založených na výsledcích měření výsledným programem. V rámci implementace dojde pouze k omezené konkrétní implementaci a to pro procesory s architekturou x86 od společnosti Intel a operační systém Linux. Nicméně obecná část implementace bude univerzální pro všechny architektury a operační systémy.

Kapitola 2

Architektura procesoru a paměti

2.1 Procesor

Procesor, též označovaný jako CPU (Central Processing Unit), je klíčovou součástí každého počítače, jelikož vykonává programové instrukce aktuálně bežícího programu. Jedná se o součástku s jednou z největších spotřeb elektrické energie v celém počítači, kterou předčí už jen vysokovýkonové grafické karty a akcelerátory. Zároveň zásadně ovlivňuje rychlost celého počítače, protože téměř vše v počítači řídí.

Konstrukčně je procesor elektronický obvod založený na logických branách, které provádějí základní boolovskou logiku [13], tedy operace AND, OR a NOT. Logická brána je složena z několika tranzistorů a jejich množství a funkčnost je určena výrobní technologií. V dnešní době je v drtivé většině procesorů použita technologie CMOS (Complementary Metal Oxide Semicoductors), která umožňuje nízkou spotřebu elektrické energie oproti předchozím technologiím, ale vyžaduje dvojnásobné množství tranzistorů, které jsou ale levné.

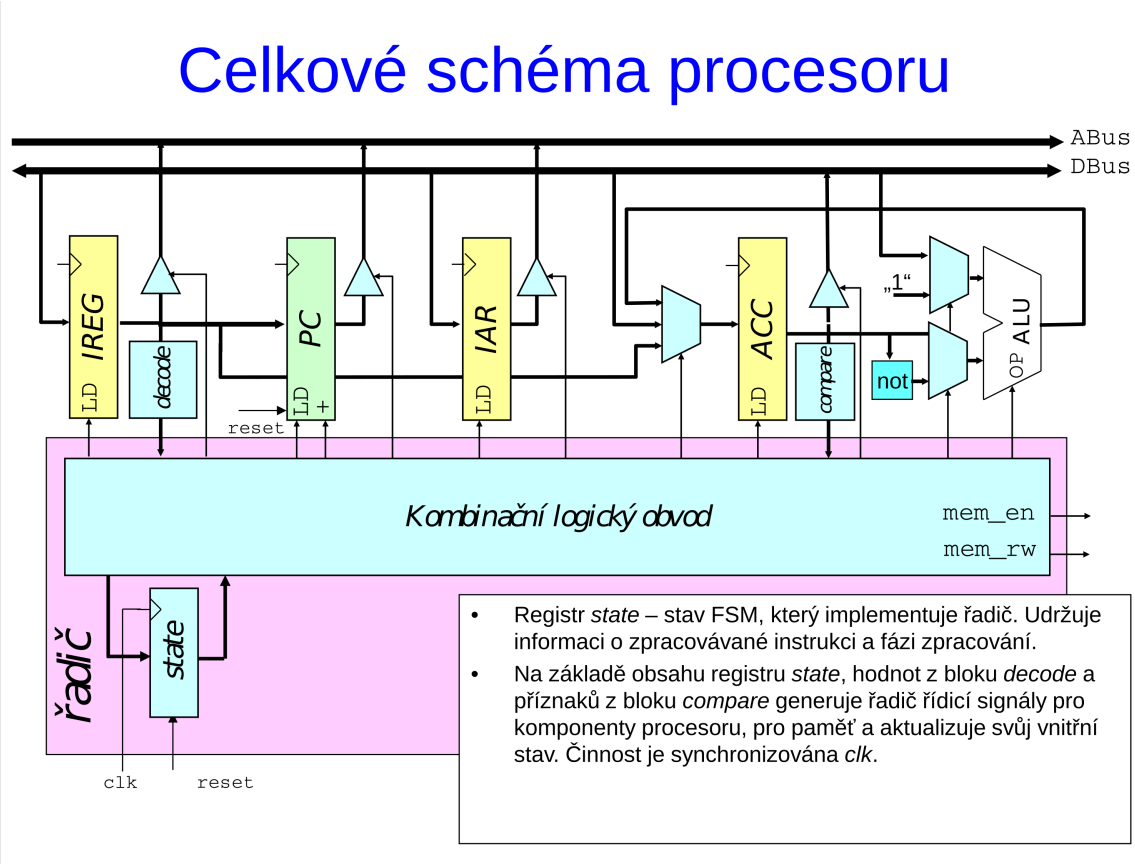
Procesor lze vujádřit jako blokové schéma komponent, které vykonávájí svou činnost a komunikují s ostatními částmi procesoru. Mezi tyto bloky patří například řadič paměti, instrukční čítač, instrukční registr, dekodér instrukcí, aritmeticko logická jednotka a další. Na obrázku 2.1 lze tato bloková schémata vidět.

2.2 Instrukční sady procesorů

Aby mohl takový procesor vůbec fungovat potřebuje mít instrukční sadu, která by obsahovala instrukce k provádění všech operací, které procesor umí zpracovat. Takovýchto instrukčních sad existuje mnoho a je potřeba, aby byl program pro takový procesor napsán v jeho instrukční sadě. Velmi rozšířené instrukční sady jsou například x86 (Intel a AMD) a ARM (ARM), své zastoupení na trhu ale najdou i instrukční sady procesorů PowerPC (IBM) a dalších.

x86

Instrukční sada x86 byla vytvořena společností Intel a jejím cílem je poskytovat vysoký výkon i za cenu vyšší spotřeby a používá se v osobních počítačích a serverech.



Obrázek 2.1: Základní blokové schéma procesoru (převzato z [10]).

x86–64

Jedná se o rozšíření instrukční sady x86 o 64 bitové registry a instrukce s nimi. Procesory vybavené touto instrukční sadou jsou tedy zpětně kompatibilní s těmi, které používají instrukční sadu x86. Tato instrukční sada obsahuje instrukce, které jsou typu CISC [25] a jako takové mohou přistupovat rovnou k paměti.

ARM

Oproti tomu instrukční sada ARM, vytvořená společností ARM holdings, byla navržena k nízké spotřebě a je tedy využívána v mobilních zařízeních. Instrukční sada je typu RISC [25], která nemá instrukce schopné pracovat přímo s pamětí [12] a tudíž jsou potřeba minimálně dvě instrukce a to uložení z paměťi do registru a instrukce pracující s registrem.

2.3 Významné registry

MSR

MSR registry procesoru s instrukční sadou x86 obsahují informace o spotřebované energii z jader procesorů, RAM modulů, nevýpočetních částí CPU, apod. Díky těmto registrům je možné zjistit, kolik energie procesor spotřeboval například v průběhu vykonávání nějakého

algoritmu. Tyto údaje pocházejí z technologie RAPL (Running average power limit) [28] od společnosti Intel, ale ostatní výrobci procesorů s instrukční sadou x86 mohou mít podobnou technologii jiného názvu. Ke čtení z těchto registrů slouží instrukce RDMSR [19].

Performance monitor counter

Jedná se o speciální čítače sloužící k zaznamenávání událostí procesoru. Některé architektury spoléhají pouze na obecné čítače a jiné mají i fixní čítače, jelikož se jedná o často požadovanou informaci. Například se takto dají zjistit informace o množství vykonaných instrukcí a skutečný počet cyklů procesoru.

Práce s obecným čítačem je popsatelná jako zapsání hodnoty určující typ události do řídícího registru čítače, jeho spuštění a jeho pozdější přečtení nebo očekávání vyvolání přerušení v případě, že hodnota přeteče velikost čítače.

2.4 Frekvence procesoru

Frekvence procesoru zásadně ovlivňuje jeho výkonnost, spotřebu energie a množství tepla, které je potřeba odvést jinam. Z tohoto důvodu se frekvence procesoru přestala navyšovat a začal se zvyšovat počet jader procesoru, tedy více procesorů v jednom čipu. Díky této možnosti paralelizace se frekvence začala snižovat zvláště u serverových procesorů.

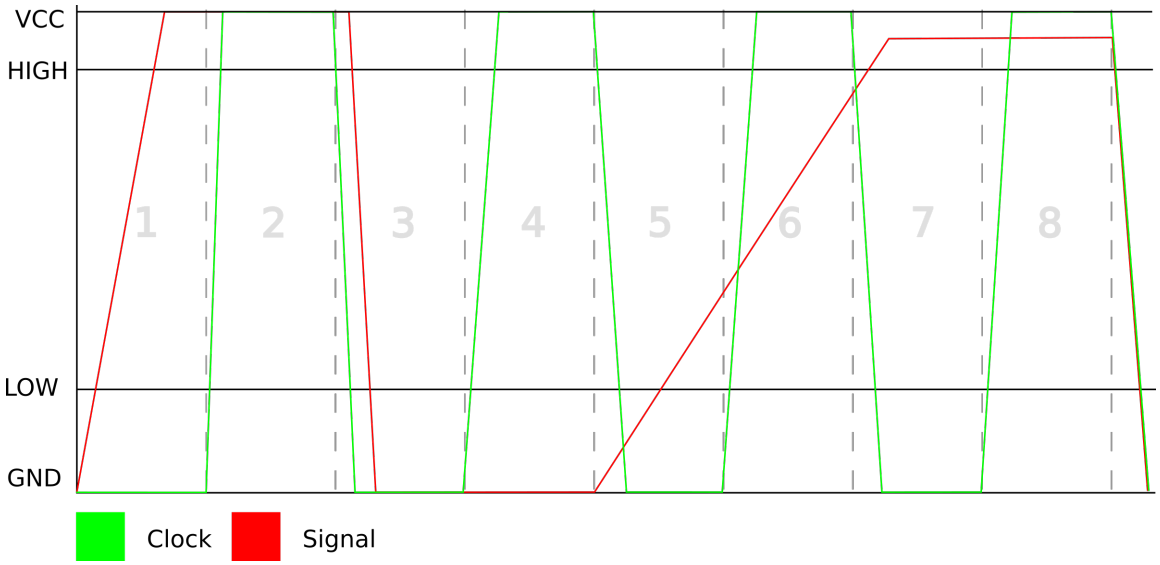
Jak bude upřesněno při popisu pamětí, narůstající frekvence procesoru efektivně zvyšuje pouze spotřebu energie a množství vyprodukovaného tepla, takže její zvyšování je tak bezpředmětné ve většině případů, ale existují i případy, kdy je vyšší frekvence lepším řešením než paralelizace. Takovýmto případem může být například práce s malým množstvím dat, které musí být zpracováno a odesláno dál, co nejrychleji.

Frekvenci procesoru lze v dnešní době ovládat třemi způsoby. Prvním způsobem je nastavení základní frekvence a výběr frekvenčního násobiče v BIOSu, ale tento způsob neumožňuje nastavit frekvenci na žádost, jelikož se počítač musí restartovat pro nastavení nové frekvence. Druhým způsobem je automatické nastavování frekvence podle vytíženosti a teploty procesoru. Tento způsob spoléhá na vytíženost procesoru, což nelze jednoznačně určit, protože každá operace trvá rozdílný počet cyklů a vyžaduje třeba jinou výkonnou jednotku. Je tedy na výrobci procesoru podle čeho se bude řídit a může tím být procesor zbytečně taktován příliš rychle pro chtěnou činnost. V tomto případě se mění pouze násobič a základní frekvence zůstává stejná. Tento způsob ale zároveň umožňuje krátkodobé překročení běžné frekvence (Intel turboboost), pokud má procesor dostatečné chlazení. Třetím způsobem je ruční nastavení výsledné frekvence, tedy výběr určitého násobiče frekvence. Tímto způsobem můžeme experimentálně zjistit jaká frekvence na daném procesoru bude poskytovat nejvyšší potřebný výkon při zachování nižší spotřeby.

2.5 Spotřeba energie

Spotřebu elektrické energie způsobuje odpor, přes který prochází elektrický proud. V procesoru je takovým odporem tranzistor, který není přepnut ani do jednoho ze dvou možných stabilních stavů (technologie CMOS [24]), tedy je ve stavu přepínání se do jednoho z nich.

Rovnice pro výpočet výsledného výkonu $P = U * I$ [33] nám ukazuje, že výkon je určen jak napětím, tak proudem, který prochází obvodem. Tento vzorec však není úplně přesný, protože považuje procesor vždy ve stavu přechodu, proto je vhodné přidat do vzorce koeficient času v přepínání vůči času v klidovém stavu, který je ovlivňován frekvencí tranzistoru.



Obrázek 2.2: Nestabilita procesoru díky vysoké frekvenci. V časovém okně číslo 6 můžeme vidět, že signál ještě nedosáhl požadované úrovně během aktivního hodinového signálu. Tento signál byl tím pádem chybně vyhodnocen jako logická 0.

Spotřeba se tedy odvíjí od rychlosti přepínání těchto stavů, tedy frekvence procesoru. K udržení růstu frekvence musí být přechodová hrana dostatečně rychle přepnuta ze stavu log. 0 do stavu log. 1. a obráceně. Toho lze se zvyšující se frekvencí přepínání docílit pomocí zvýšení napětí pro tranzistory. Pokud není napětí dostatečně zvýšeno, může dojít k nestabilitě procesoru. Tato nestabilita je způsobena tím, že hodinový signál se přepne dříve než dojde k přepnutí z logické 1 na logickou 0 a obráceně dostatečným nárůstem napětí na tranzistoru. Toto chování je demostrováno na obrázku 2.2.

Ohmův zákon ale hovoří jasně, pokud se zvýší napětí v obvodu (viz rovnice 2.1 a 2.2), pak se zákonitě zvedne i spotřeba (viz rovnice 2.3 a 2.4), jelikož se zvýší množství proudu v obvodu, což vede ke zvýšení spotřeby nejen díky vyššímu napětí, ale i vyššímu proudu.

$$I_1 = U_1 * R \tag{2.1}$$

$$I_2 = U_2 * R \tag{2.2}$$

$$P_1 = U_1 * I_1 \tag{2.3}$$

$$P_2 = U_2 * I_2 \tag{2.4}$$

Spotřeba se zvedá i pouze se zvyšováním frekvence přepínání bez zvýšení napětí, jelikož dochází k častějším přepnutím a tedy častěji dochází k odporu při přepínání. Toto lze demonstrovat pomocí koeficientů N_1 a N_2, které vyjařují poměr času stráveného v přepínacím stavu vůči celkovému času, kde koeficient N_2 znamená vyšší frekvenci přepínání.

$$N_1 = 0.2 \tag{2.5}$$

$$N_2 = 0.3 \tag{2.6}$$

$$P_1 = U * I * N_1 \tag{2.7}$$

$$P_2 = U * I * N_2 \tag{2.8}$$

2.6 Teplota

Naprostá většina energie, která je procesorem spotřebována, se mění na teplo, protože se procesor stává odporem. Toto teplo musí procesor opustit, jinak by došlo ke zvýšení teploty a přehřátí. Vnitřně se teplo přenáší kovovými vodiči, tedy část tepla opouští procesor jeho vývody na základní desce. Větší část tepla musí proceosor opustit přes jeho povrch a proto se na povrch samotného čipu přidává takzvaný heatspreader [23].

Vysoká teplota fyzikálně způsobuje ještě vyšší odpor [15] a tudíž je chtěné, aby byla teplota co nejmenší. Zároveň by vysoká teplota mohla způsobit zkreslení dat kvůli falešnému sepnutí tranzistoru z důvodu snížení potřebného napětí k jeho sepnutí [30]. Ke stejnému negativnímu výsledku by mohlo dojít z důvodu malého množství propuštěného proudu způsobeného vysokou teplotou a následující tranzistor by se nemusel otevřít [30].

Ke sledování teploty jsou v procesoru zabudována teplotní čidla a jejich hodnoty je možné číst ze speciálních registrů procesoru. Takováto čidla jsou umístěná v každém jádře a na povrchu procesoru. Hodnoty těchto registrů slouží jak pro vnitřní účely procesoru, viz podkapitolu 2.4, tak se dají číst v uživatelském prostředí například pomocí periodického zápisu do souboru.

2.7 Výkonnost

Výkonnost lze měřit pomocí mnoha měřítek, nejčastějším měřítkem je však počet float-point operací za jednu sekundu. Dnes se pohybujeme v řádu jednotek až desítek petaFLOPů za sekundu při pohledu na superpočítače a v řádu stovek gigaFLOPů za sekundu až po jednotky teraFLOPů za sekundu při pohledu na běžnou výpočetní elektroniku, jako jsou grafické karty.

Obecný vzorec pro výpočet FLOP/s je definovaný rovnicí 2.9. Nicméně toto je jen hrubý odhad, jelikož se nedá odhadnout přesný výkon, protože záleží na tom, zda se využijí SIMD (instrukční sady SSE nebo AVX) jednotky procesoru nebo se použije FPU (instrukční sada x87).

$$\frac{FLOP}{sekunda} = počet\ CPU \times \frac{počet\ jader}{procesor} \times frekvence \times \frac{počet\ operací}{takt} \tag{2.9}$$

Výkonnost superpočítačů můžeme nalézt seřazenou podle největšího výkonu například v žebříčku Top500 [8], který obsahuje nejvýkonnější superpočítače z celého světa. Lze v něm nalézt informace jako je počet procesorových jader a grafických karet společně s jejich maximálním výkonnem v GFLOPSech. Tento žebříček je aktualizován přibližně každého půl roku.

2.8 Energetická efektivita

Problémem energetické efektivity se zabývá mnoho vědců a výrobců procesorů po celém světě. Pokud by efektivita nebyla řešena, tak by došlo k neřešitelným problémům jako je příliš mnoho tepla z tranzistorů a tudiž by se zastavil nárůst výkonu procesorů. Efektivita vybraného čipu se dá jednoduše spočítat podělením počtu GFLOP/s a příkonu čipu. Efektivita se bude měnit s frekvencí čipu, ale zároveň se mění i příkon. Je tedy potřeba uvést při jaké frekvenci a napájecím napětí se čip nacházel v době výpočtu této hodnoty. Například: $1,541\,GFLOP/s/W$ při frekvenci $1200\,MHz$ a napájecím napětí $1,15V$.

Efektivita čipu se dá zvýšit několika způsoby. Mezi ně patří snížení napájecího napětí, použití více jader na nižší frekvenci a modernizace čipu. Snížením napětí se zmenší příkon čipu. Toto lze aplikovat jen do chvíle než čip začne být nestabilní. Tato nestabilita je způsobena nedostatečným napájením, kdy nelze rozlišit o jakou logickou hodnotu se jedná.

Některé algoritmy lze paralelizovat a je proto jednodušší raději použít vice jader na nižší frekvenci, než zbytečně vytěžovat jedno jádro na vysoké frekvenci. Nižší frekvence vyžaduje menší napájecí napětí a tranzistory budou méně často ve stavu mezi logickou jedničkou a logickou nulou, což povede ke zmenšení odporu v procesoru.

Další možné řešení je vniřní modernizace čipu zahrnující např. dvojí sadu registrů pro každé jádro (technologie HyperThreading), kdy při čekání na naplnění dat do první sady je možné pracovat se sadou druhou nebo zpracovat obě naráz, pokud nepožadují stejné části procesoru (např. sečtení celých čísel a dělení v plovoucí čárce) a jsou obě již načteny.

Efektivitu superpočítačů lze nalézt například v žebříčku Green500, který ukazuje efektivnost superpočítačů v žebříčku TOP500 pomocí FLOPSů na watt příkonu celého superpočítače. Je zajímavé, že superpočítač Sunway TaihuLight [31] z prvního místa v žebříčku TOP500 v listopadu 2017 se nachází na dvacátém místě žebříčku Green500. Jeho efektivnosti bylo dosaženo vlastní architekturou procesoru, který je typu RISC a má 260 jader o nízké frekvenci [17]. Více zajímavé je například to, že ho v energetické efektivnosti předběhly superpočítače Gyoukoua a Piz Daint využívající GPU, které jsou v žebříčku Top500 na čvrtém a třetím místě. Je zde tedy vidět jasný pokrok jak ve výkonu celého systému, tak v jeho energetické efektivnosti.

2.9 Měření spotřebované energie

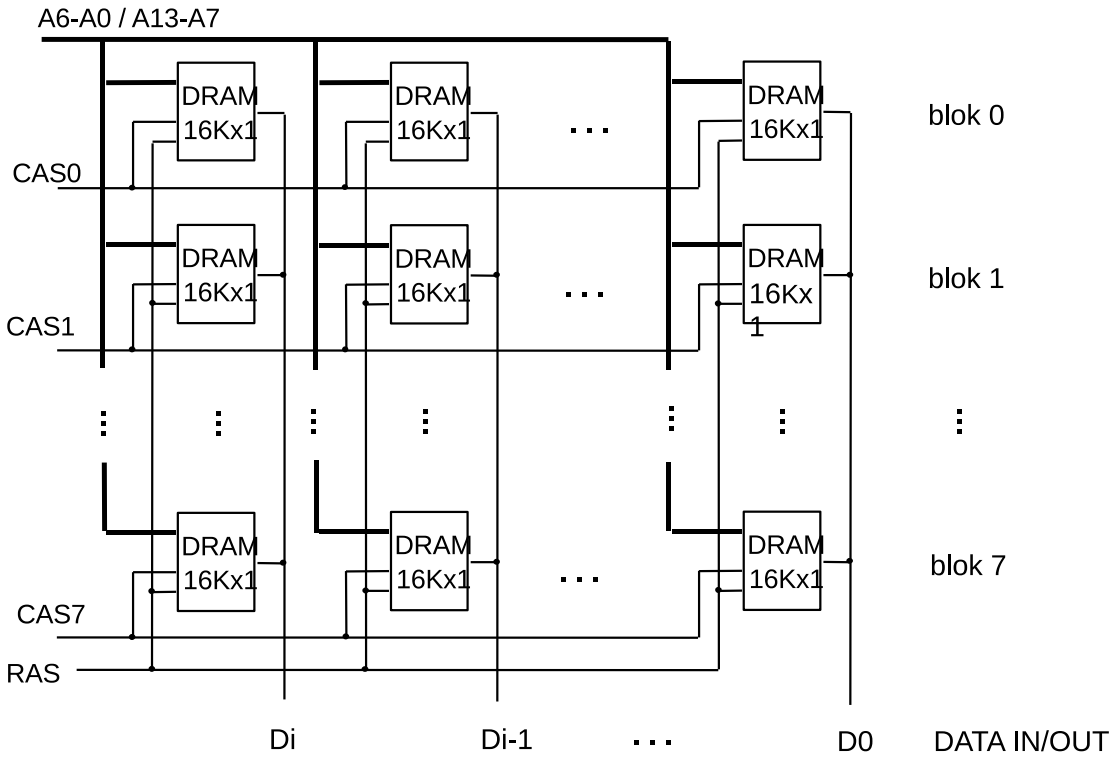
Nejjednodušší by bylo, pokud by se nám povedlo zapojit zařízení takzvaný wattmetr [22] mezi základní desku a samotný procesor. To bohužel není často možné, protože server se může nacházet v zabezpečeném serverovém středisku, a proto výrobci procesorů přišli s vlastním řešením, které využívají v případě technologií úspory (Intel Speedstep, AMD Cool&quiet) nebo naopak v nárůstu výkonu (Intel Turboboost) ve chvílích, kdy je to požadováno a z hlediska napájení a chlazení možné. V praxi se tak vytváří všechny možné měřiče a čítače, které jsou integrované do samotného procesoru.

Mezi takové čítače patří PMU (Performance monitor unit), kde nás zajímá technologie Intel RAPL (Running average power limit [28]) a MSR registry, kam se výsledky zapisují. Přístup k těmto čítačům je možný několika způsoby. Mezi ně patří převedení obsahu registrů MSR do uživatelského prostoru systému, respektive periodický zápis do virtuálního souboru, což ale vyžaduje, aby systém měl nahraný modul MSR. Další možností je využití technologie RAPL, která je implementována pouze v procesorech společnosti Intel.

2.10 Paměť

V paměti RAM jsou uloženy instrukce a data určená ke zpracování. Paměť RAM narozdíl od procesoru funguje na nižších frekvencích a tudíž procesor může být zpožděn čekáním na potřebná data z paměti RAM. Tomuto se dá zabránit implementací paměti cache, která bude popsána později.

Dnešní procesor pracuje na frekvencích okolo $3000MHz$. Paměť RAM pracuje na frekvencích $1000 - 2000MHz$. Je tedy zřejmé, že zpoždění dat bude minimálně 3 cykly. Tyto



Obrázek 2.3: Maticové uspořádání paměti RAM (převzato z [11]).

výsledky je potřeba považovat pouze za ukázku, realné zpoždění je daleko vyšší. Nicméně existují technologie, které zvětšují množství přenesených dat a budou popsány později.

Dnes používaný standard pamětových modulů DIMM přenáší naráz 64 bitů dat. Aby bylo možné k těmto datům přistoupit, je potřeba pochopit, jak jsou data v paměti uložena. Pokud bychom chtěli mít každou paměťovou buňku adresovanou přímo, pak bychom potřebovali mít enormní množství adresových vodičů, což není technologicky možné provést, proto paměti využívají uspořádání paměťových buněk do matice (viz obrázek 2.3). Uspořádání do matice ale přináší nový problém adresace a to je 2D adresace, je tedy potřeba nejdříve vybrat řádek a poté sloupec. Každá z těchto operací není okamžitá a je na ni potřeba nějaká doba. Tato doba je uváděna jako časování paměťového modulu.

Jedním z řešení, jak zrychlit tento proces, je technologie DDR, která umožňuje přenést informace jak na nástupní hraně hodinového cyklu, tak na hraně sestupní. Efektivně tedy zdvojnásobí množství dat přenesených během jednoho hodinového cyklu. Další z technologií je kanálování pamětí, kdy místo výběru paměťového modulu jsou moduly adresně poskládány vedle sebe a tím pádem vzniká širší sběrnice. Na příkladu dvou-kanálového adresování můžeme vypočítat, že šířka sběrnice bude 2×64 bitů, tedy 128 bitů.

Těmito technologiemi jsme obešli technicky obtížné adresování paměti. Nicméně v tuto chvíli je paměť ještě pomalejší, protože se musí vypočítat finální adresa při každém dotazu. Řádově je zpomalení asi desetinásobné. Jako kompenzace tohoto zpomalení byla vytvořena paměť cache, která je zabudována v procesoru a je prakticky stejně rychlá jako samotný procesor. Její velikost se postupem času zvyšovala a přidávaly se další úrovně, jako je možné vidět v tabulce 2.1. Zajímavé na této tabulce je, že velikost L1 a L2 cache paměti se od roku

Tabulka 2.1: Velikost cache pamětí procesorů v průběhu historie. Zdroj dat: [2]

Procesor	**Rok uvedení na trh**	**L1**	**L2**	**L3**
Intel 486DX	*1989*	8KB	X	X
Intel Pentium 4 641	*2006*	16KB	2048KB	X
Intel Atom 230	*2008*	32KB/24KB	512KB	X
Intel Xeon E5-2620	*2012*	6x32KB/6x32KB	6x256KB	15MB
Intel Core i5-4200m	*2013*	2x32KB/2x32KB	2x256KB	3MB
Intel Xeon E7-4850 v4	*2016*	16x32KB/16xKB	16x256KB	40MB

2012 u procesorů společnosti Intel nezvětšila. Do této paměti se data musí dostat z paměti RAM, tedy při prvním dotazu na data bude procesor několik taktů nečinný, respektive bude vykonávat jiné vlákno. Ve chvíli, kdy jsou data načtena, tak se zpracují a požadují se další data, která jsou v paměti hned vedle a zde přichází výhoda paměti cache a to ta, že tyto data už pravděpodobně byla načtena také, jelikož jsou v paměti hned vedle. Tímto dojde k výraznému urychlení práce s daty a procesor nečeká na paměť RAM tak často.

Kapitola 3

Využití existujících nástrojů a možná vylepšení

V této práci budou využity již vytvořené programy a zpracují se jejich výstupy. Tento přístup by měl zajistit, že využité programy jsou již otestovány jejich vývojáři a tudíž je malá šance, že by zanesly chyby do výsledné sady skriptů, protože vývoj takzvaně „od nuly“ je vždy riskantní.

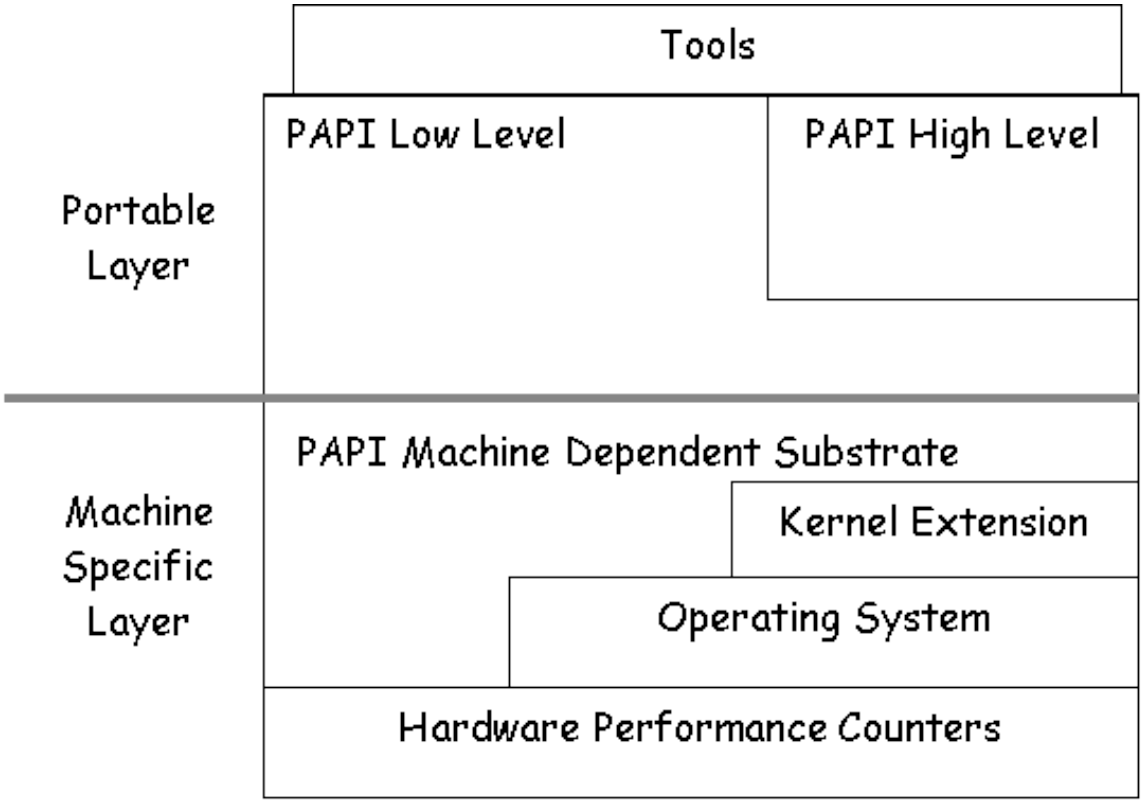
Druhým důvodem pro využití již hotových programů je obrovská modularita při sledování spotřeby. K těmto programům je zapotřebí dopsat pouze druhou stranu a tou je parser výstupu daného programu, což je značně jednodušší než celý vývoj. Zároveň tento způsob umožňuje sledovat spotřebu již napsaných aplikací a vyladění procesoru pro jejich běh bez zásahu do kódu aplikace. Toto se týká především spotřeby energie, ale jiná měření, například události pomocí knihovny PAPI, vyžadují zásah do kódu zároveň s napsáním požadovaného parseru. Tímto je výsledná sada skriptů schopná pokrýt jak již hotové programy a pomoci s jejich optimalizací pro daný server, tak i vývoj nových aplikací společně s jejich optimalizací na úrovni algoritmu.

Kromě spotřeby jednotlivých částí CPU je při optimalizaci algoritmu potřeba znát vlastnosti algoritmu jako například počet vykonaných instrukcí vůči počtu taktů procesoru. Tyto hodnoty lze získat pomocí registrů PMC (viz kapitolu 2.3) po jejich naprogramování. Tuto činnost provádí například knihovna PAPI (viz kapitolu 3.1). V uživatelském prostředí o tomto měření hovoříme jako o čítači událostí (event counter). Práce s těmito čítači je poté velmi jednoduchá, jelikož se pouze spustí před částí sledovaného kódu a na konci této části se čítače zastaví a přečtou se výsledná data z registrů PMC.

3.1 PAPI

Performance API [29], známé také jako PAPI, je sada multiplatformních knihoven poskytujících přístup k takzvaným event counterům [6]. Jedná se o nejpoužívanější knihovnu, jelikož její naportování na potřebnou architekturu není složité, pokud platforma podporuje požadované možnosti monitorování.

Toto API se skládá ze dvou vrstev knihoven. Prvni vrstva je portabilní na nové platformy jen s malými úpravami a kompilací na nové architektuře. V této vrstvě se nachází high level a low level API, která budou popsána dále v textu. Druhou vstvu tvoří sada knihoven, takzvaný substrát, a je specifická pro každou architekturu, jelikož obsahuje přístup přímo k hardwaru této architektury.



Obrázek 3.1: Struktura Performance API s dělením na přenosnou uživatelskou a architekturně závislou část (převzato z [5]).

High-level API umožňuje pouze nastavení sledovaných událostí, spuštění sledování, zastavení sledování a získání výsledků. Tento přístup nám pro použití v této práci stačí, jelikož se zabývá pouze zpracováním výsledků.

Low-level API umožňuje programátorovi navrhnout vlastní událost a jeho zpracování. Na této úrovni lze například vytvořit callback funkci, která bude spuštěna v případě přetečení registru a měla by na tuto událost reagovat. Touto úrovní se ale tato práce nezaobírá, nicméně při dodržení stejného výstupu jako z high-level API je program schopný zpracovat i výsledky z této úrovně.

3.2 Intel Performance Counter Monitor

Intel Performance Counter Monitor [4] je sada nástrojů velmi podobná PAPI. Narozdíl od PAPI umožňuje sledovat informace specifické pouze pro procesory od společnosti Intel. Takovouto informací je například vytížení sběrnice QPI, která se nachází v procesorech od této společnosti.

Stejně jako PAPI, tyto nástroje pracují s registry MSR a RAPL a poskytují jak informace o výkonnostních parametrech procesoru, tak informace o spotřebě jednotlivých částí procesoru.

3.3 Vylepšení poskytovaná v rámci této práce

Unifikovaný sběr informací

Program, který je výsledkem této práce má jednotné rozhraní pro získání a uložení informací. Je tedy možné ho jednoduše rozšířit o jakýkoli nový program, který získává informace o běhu sledovaného programu.

Tato data jsou uložena v univerzálním formátu v databázi. Formát dat umožňuje uložení informace s potřebnými označeními času a jádra procesoru, ze kterého tato informace pochází. Je tedy například možné vytvořit histogramy jednotlivých měření.

Dlouhodobé uchování výsledků s možností porovnání

Všechna měření jsou uchovávána v databázi pro pozdější zpracování. Je tedy možné vytvořit grafy porovnávající například spotřebu před a po úpravě algoritmu o měsíce později. Tato funkcionalita je nesmírně důležitá pro regresní testování a proto bylo zvoleno ukládání do databáze jako nejvhodnější způsob pro udržení relevantních měření pohromadě.

Výpis výsledků do grafů

Jeden obrázek vydá za tisíc slov, v případě grafu možná i za dva tisíce. Zobrazení výsledků v grafech je daleko pohodlnější pro čtení než posloupnosti čísel, která program naměří. Většina programů je dělána pro jednoduché využití dalšími a tedy funkcí vykreslování grafů je vybavena až tato sada skriptů.

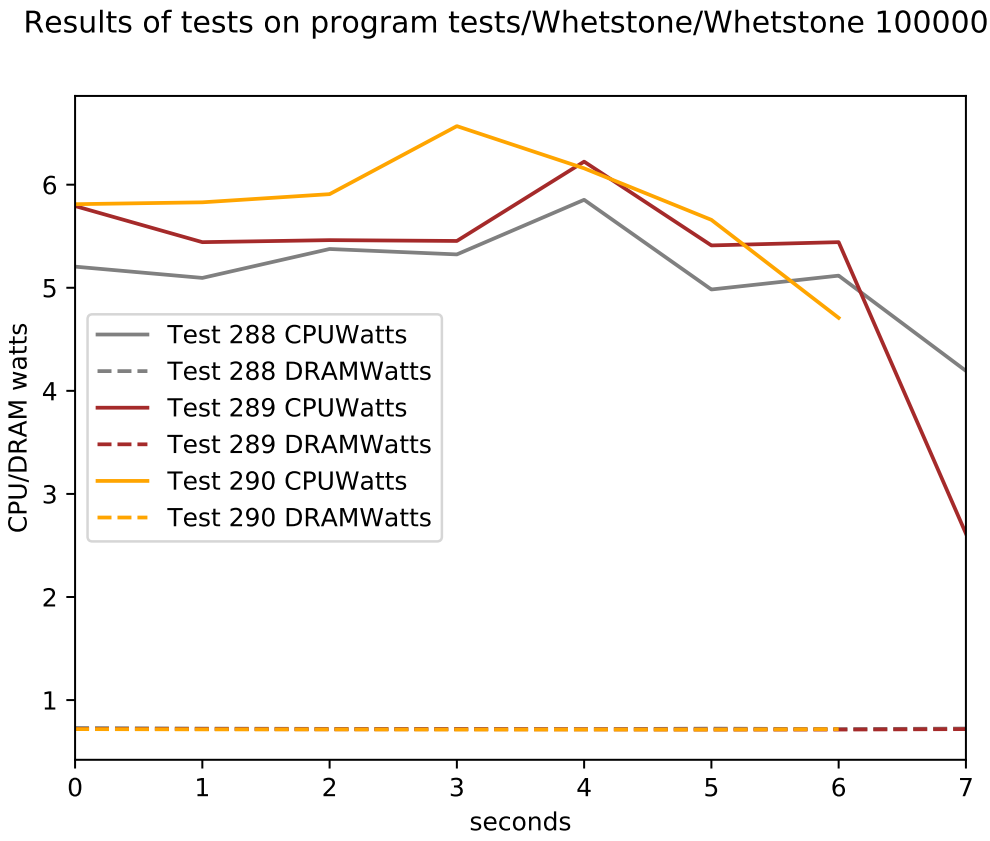
K vykreslení grafu je vhodné využít již existující nástroje určené k vykreslování grafů. Takovým nástrojem je knihovna matplotlib, která byla využita v rámci této práce, jelikož poskytuje jednoduché funkce pro práci s velkým množstvím dat a jejich průběžným zakreslováním do grafu, což lze vidět na obrázku 3.2. Tato knihovna je sama napsána v programovacím jazyce Python a proto byla jasnou volbou v rámci této práce, kvůli shodnému implementačnímu jazyku.

Rozšířitelnost o moduly uživatele

Hlavním cílem této práce je představit sadu skriptů, která automaticky zpracovává vstupy z využívaných podpůrných programů, avšak nejednalo by se o kvalitní práci pokud by byla omezena přesně na to, co bylo zadáno a nic jiného by neuměla a nikdy nemohla umět. Vedlejším cílem se tedy stala rozšířitelnost s minimálním zásahem uživatele, kterého by aktuální implementace omezovala.

Program je rozšířitelný téměř ve všech jeho částech. Od začátku byl navrhován tak, aby byl rozdělen do funkčních bloků, které vždy obsahují rozhraní pro prácí s jakýmkoliv skriptem v daném bloku, který implementuje toto rozhraní.

Jako příklad lze uvést práci s databází, jelikož program pracuje s obecným skriptem pro práci na vyšší úrovni, jako například zvýšení hodnoty v tabulce databáze na řádku, který vyhovuje podmínce. Tento skript pracuje s obecným rozhraním databáze, které tím pádem musí tuto funkcionalitu podporovat a skript pro konkrétní databázi ho již implementuje v dotazovacím jazyku databáze. Součástí práce je skript pro databázi SQLite3, která používá jazyk SQL, avšak pouze jeho omezenou podobu a proto je potřeba mít rozdílné skripty pro databáze SQLite, MySQL a další. Každý tento skript bude danou funkcionalitu podporovat ve verzi dotazovacího jazyka, který je podporován danou databází. Jelikož byla



Obrázek 3.2: Návrh výsledného grafu generovaného modulem pro výstup dat do grafů

práce s databází velmi dobře izolována je klidně možné naprogramovat skript, který se na venek jako databáze chová, ale jeho implementace může být jakákoliv.

Kapitola 4

Návrh řešení a implementace

Ze zadání práce vyplývá, že výsledkem musí být sada skriptů, které mají být schopné posbírat data o spotřebě procesoru a dalších možných parametrech sledovaného programu a získaná data interpretovat do podoby grafu nebo tabulky. Rozdělením na jednotlivé skripty lze dosáhnout flexibility pro případ použití, kdy dochází ke sběru dat na serveru a jejich interpretaci až na lokálním počítači.

Jako jazyk k implementaci byl v zadání doporučen jazyk Python a proto byly skripty implementovány v tomto jazyce. Jelikož se jedná o velmi rozšířený skriptovací jazyk, neměly by nastat problémy s nasazením na existující servery a superpočítače.

Celá sada skriptů dodržuje paradigma objektově orientovaného programování a proto lze tedy definovat rozhraní, pomocí kterých tyto skripty spolupracují a je možné jednoduše použít jiný skript, který dodržuje toto rozhraní.

Pro vývoj a testování těchto skriptů byla použita architektura x86 a procesory od společnosti Intel, jelikož jsou použity ve 457 z 500 nejvýkonnějších serverů podle žebříčku Top500 z listopadu 2017 [9] a přínos této práce je určen právě pro ně. Rozšíření těchto skriptů na jiné architektury není v součástí této práce, avšak skripty jsou navrhovány na jednoduché rozšíření na jakoukoli architekturu.

Sada skriptů se skládá z těchto modulů:

1. Jádro
2. Databáze
3. Vykonavači
4. Monitory výkonu
5. Výstupní modul
6. Ovládání frekvence procesoru
7. Modifikace běhového prostředí sledovaného programu

4.1 Návrh modulů

Jádro

Jedná se o centrální modul této práce, bez kterého by ostatní moduly neměly možnost korektně fungovat, jelikož tento modul kromě jiného spouští ostatní moduly. Jednou z jeho

funkcí je načtení a zpracování vstupů od uživatele, čímž odstiňuje ostatní moduly od specifických informací od uživatele, které daný modul nepotřebuje.

Databáze

Modul pro abstraktní práci s databází, který odstíní uživatele i tvůrce modulů od implementace databázových přípojek. Součástí modulu by mělo být obecné rozhraní práce s jakoukoli databází a minimálně jeden určitý způsob připojení k databázi.

Vykonavači

Vykonavači slouží k vykonání sledování zadaného programu. Je možné vykonat pouze jeden program nebo celý dávkový soubor.

Dávkový soubor je specifický pro tuto sadu skriptů, protože umožňuje při velmi malém počtu znaků specifikovat různé argumenty pro program ke sledování. Tento soubor je tvořen řádky s jednotlivými programy ke spuštění a sledování, kde každý řádek obsahuje jméno programu a skupiny argumentů nebo komentář. Použitím více skupin vzniknou kombinace všech možných parametrů zadaných ve skupinách. Skupina argumentů se zadává jako seznam argumentů oddělených mezerami zabalených do hranatých závorek, přičemž argumenty uvnitř takové skupiny jsou vzájemně výlučné. Je možné zadat hodnoty těchto argumentů a to buď řetězec znaků, seznam hodnot nebo řetězců nebo rozsah číselných hodnot, kde je možné specifikovat velikost kroku. Dalším velkým přínosem je možnost zadání ignorování jedné možnosti ze seznamu dané skupiny, což způsobí, že se daná skupina bude ignorovat, což je například vhodné, pokud bude použito více těchto skupin. Pro větší pochopení je vhodné uvést příklad použití dávkového souboru 4.1 a jeho výsledek 4.2.

```
1 gcc[# -o1 -o3][werror pedantic][-std"c99"][#"lorem.c"]
```

Výpis 4.1: Příklad více skupin dávkového souboru s použitím znaku pro ignorování dané skupiny

```
1 gcc --werror -std=c99 lorem.c
2 gcc --pedantic -std=c99 lorem.c
3 gcc -o1 --werror -std=c99 lorem.c
4 gcc -o1 --pedantic -std=c99 lorem.c
5 gcc -o3 --werror -std=c99 lorem.c
6 gcc -o3 --pedantic -std=c99 lorem.c
```

Výpis 4.2: Výsledek dávkového souboru 4.1

Monitory výkonu

Tento modul se stará o klíčovou funkcionalitu této práce, jelikož jeho prostřednictvím se lze napojit na externí meřicí programy, případně číst výsledná data ze souborů. Součástí tohoto modulu je rozhraní obecného chování pro implementaci nových monitorů a minimálně jeden určitý monitor. Všechny monitory budou zapisovat do databáze pomocí již specifikovaného rozhraní pro práci s databázemi.

Výstupní modul

Přečte výsledky z databáze pomocí modulu databáze a předá je vybranému vykreslovacímu nebo výpisovému modulu. Tento modul je možné použít i v případě, že nejsou k dispozici

moduly pro získávání dat. Tímto lze umožnit vytvoření menší aplikace určené pouze pro zobrazování dat. Součástí modulu budou podmoduly určené k vykreslování do grafických struktur a pro výpis do formátů pro jiné programy.

Ovládání frekvence procesoru

Pro jednu z chtěných funkcí této práce je nezbytné mít možnost ovládat frekvenci procesoru za běhu operačního systému a tento modul tuto funkcionalitu poskytuje.

Modifikace běhového prostředí sledovaného programu

Jelikož tato práce vytváří pro sledovaný program nové prostředí pro běh je potřebné aby toto prostředí mohlo být modifikováno, protože klasická modifikace před spuštěním bude ovlivňovat pouze program pro sledování.

4.2 Struktura modulů

Jak již bylo zmíněno dříve v textu, tato sada skriptů byla vytvořena s dodržováním paradigmatu objektově orientovaného programování. V obrázku 4.1 lze tedy vidět, jaké skripty (třídy) implementují které rozhraní.

4.3 Sledování spotřeby

Samotné skripty pouze parsují výstup pomocných programů, jako je mnou vytvořený program, který je založený na práci s registry MSR a napsán v jazyce C, a program **pcm–power** z balíku nástrojů Performance Counter Monitor (PCM) od společnosti Intel. Z důvodu existence pomocných programů jsem se rozhodl je využít raději parsovat jejich výstup, než je znovu implementovat.

Součástí sady je kromě měřičů pro architekturu x86 také rozhraní pro implementaci obecného měřiče spotřeby jakéhokoli původu. Například by takto mohl být doprogramován i modul sledující spotřebu ARMových desek, které jsou bez možnosti interního měření spotřeby, a modul by přijímal výsledky měření po sběrnici RS–232 z externího měřiče nacházejícího se mezi deskou a zdrojem napětí.

4.4 Změna frekvence

Pro změnu frekvence procesoru se v Linuxu od jádra 2.6 [7] používa virtuální souborový systém SysFS a jeho složky a soubory na adrese **/sys/***. Tento souborový systém vystavuje řídicí struktury linuxového jádra do uživatelského prostoru jako složky a soubory. Jejich konfigurací se dají zjistit a nastavit nesčetné údaje, které by jinak byly nedostupné.

Pro změnu frekvence pro daný procesor je potřeba zapsat chtěnou frekvenci do souboru **/sys/devices/system/cpu/core/cpufreq/scaling_setspeed**, kde **core** je označení pro jádro, které chceme ovládat. Abychom toto mohli provést, je potřeba zapsat do souboru **/sys/devices/system/cpu/core/cpufreq/scaling_governor** hodnotu **userspace**, která přepne ovládání procesoru do uživatelského prostředí operačního systému. Tyto frekvence je možné získat ze souboru **/sys/devices /system/cpu/core/cpufreq/scaling_available_frequencies**. Přečtením tohoto souboru dostaneme seznam možných frekvencí pro dané jádro **core**, kde jsou jednotlivé položky odděleny mezerou.

Druhým způsobem pro změnu frekvence procesoru je použití programu **cpupower**, který umožňuje stejné chování, avšak nenutí uživatele zapisovat přímo do souborů. Tento program má velmi jednoduché ovládání a například nastavení frekvence procesoru pomocí něj můžeme vidět ve výpisu 4.3.

```
1       cpupower -c all frequency-set -f 1.5GHz
```

Výpis 4.3: Nastavení frekvence procesoru pomocí programu cpupower

Jelikož v případě **cpupower** se jedná o baliček, který nemusí být součástí operačního systému, tak jsem se rozhodl zvolit přístup přímého zápisu do virtuálních souborů, které jsou k dispozici vždy od Linuxového jádra 2.6 [7]. O toto ovládání se stará obecná třída monitorů a není tedy potřeba ji pokaždé znovu implementovat pro nový monitor spotřeby.

4.5 Ovlivnění prostředí

Prostředí umožňuje předat programu proměnnou informaci, která není známá při kompilaci programu a není možné ji předat jako argument, jelikož se například jedná o knihovnu, která je přidána do jakéhokoliv programu a ten by musel implementovat argument pro danou informaci. V tomto případě je lepší využít proměnnou v prostředí, jelikož bude mít vždy stejné jméno a dá se nastavit před během programu, který ji bude vyžadovat.

Výsledná sada skriptů má ale s prostředím zásadní problém, jelikož sledovaný program běží v jiném prostředí než řídicí skript a tedy nastavení prostředí před spuštěním sady skriptů způsobí předání proměnných pouze sledujícímu skriptu a ne sledovanému programu.

Pro předání proměnných sledovanému programu je potřeba ho s těmito nastaveními spustit. Tedy nelze nastavit prostředí předem a spouštět pouze sledované programy, ale je potřeba s nimi pokaždé dočasně nastavit prostředí pro sledovaný program. Samozřejmě je možné spustit spouštěcí skript pro sledovaný program a nastavit prostředí pomocí něj.

Takovouto proměnnou prostředí je například KMP_AFFINITY, která nastavuje přiřazení vláken procesoru sledované aplikaci. Tedy není možné provést spuštění pomocí příkazu ve výpisu 4.4, ale příkazem z výpisu 4.5.

```
1 KMP_AFFINITY=xxxxx sledovaci_skript --param1 --param2 sledovany_program
```

Výpis 4.4: Špatně zadaná proměnná do prostředí pro sledovaný program

```
1 sledovaci_skript --param1 --param2 KMP_AFFINITY=xxxxx sledovany_program
```

Výpis 4.5: Správně zadaná proměnná do prostředí pro sledovaný program

4.6 Výstup dat

Abychom mohli sadu skriptů efektivně používat, je potřeba mít velmi efektivní výstup, který nám jednoduše ukáže údaje, které hledáme. Těchto výstupů existuje mnoho, avšak v této práci se zaměříme pouze na dva z nich.

Nejužitečnější bude bezesporu výstup do tabulky, kterou můžeme později využít pro jakékoliv naše pozdější zpracování. Při tomto typu výstupu jsou data přehledně uspořádána podle svého typu do sloupečků výsledné tabulky a každý řádek představuje jedno měření.

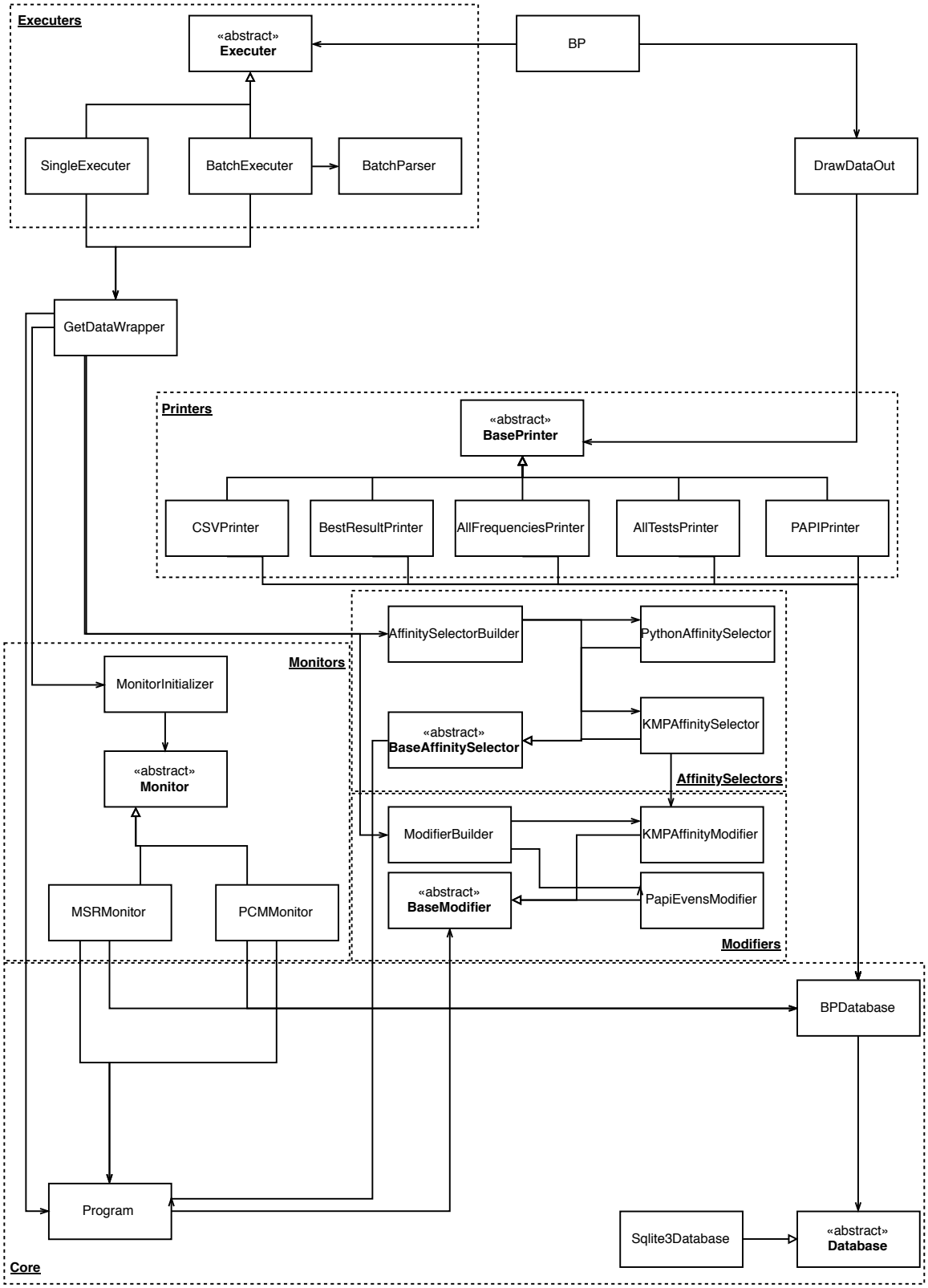
Velmi užitečné bude i grafické zobrazení v podobě grafů, kde se ukáže spotřeba v celém průběhu sledovaného programu. Na tomto průběhu bude možné sledovat události jako výkyvy spotřeby energie, které naznačují procesorově náročnější část kódu.

4.7 Výsledná implementace algoritmu

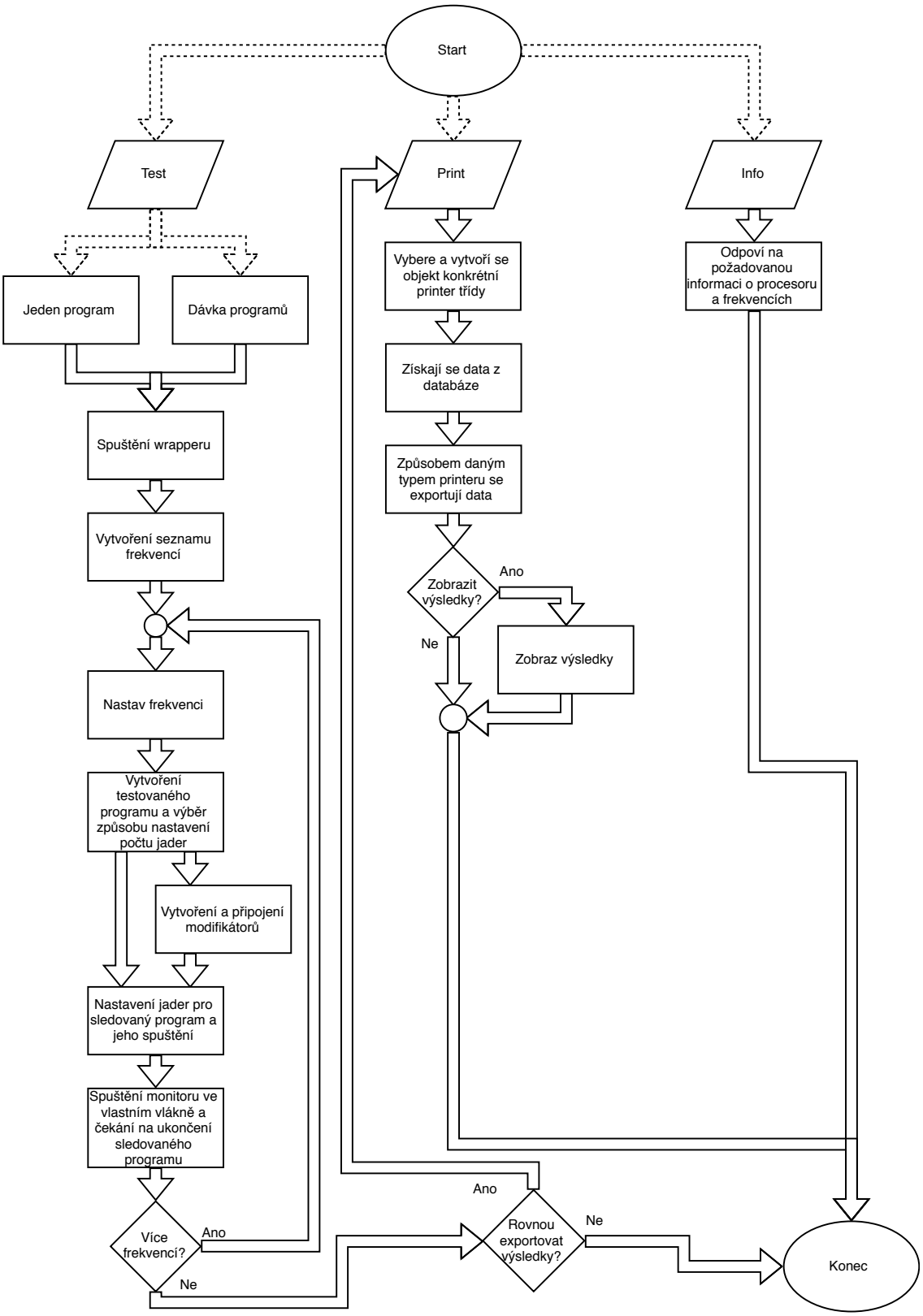
Jak můžeme vidět na obrázku 4.2, sada skriptů je rozdělena do tří samostatných sekcí a to: test, print a info. Sekce info slouží k zjištění informací o procesorech na daném počítači a zjištění pořadového čísla posledního testu.

Sekce print slouží k získání informací o zadaných testech ve formátovaném výstupu. Těmito výstupy je buď graf pro všechny frekvence s jedním z testovaných programů nebo všechny programy pro jednu z testovaných frekvencí a nebo tabulky ve formátu CSV. Automaticky dojde k vytvoření výstupního souboru. Při zadání parametru okamžitého zobrazení může v případě grafu být graf zobrazen v módu pro úpravy před finálním uložením, který zároveň slouží jako okamžitá ukázka výstupu. Při zadání tohoto parametru v případě tabulkového výstupu bude tabulka vytištěna do terminálu.

Sekce test spouští zadaný program nebo dávku programů a monitorovací program podle výběru. Dávka programů může být například použita ke spuštění více programů se stejnými PAPI událostmi bez potřeby skriptovat spuštění sledovacího programu ve vlastním skriptu. Druhé využití dávkového spuštění je možnost zadat skupiny parametrů, přes které bude sledovací program iterovat a spouštět zadaný program s parametry podle aktuální iterace. Zadáním parametrů do stejné skupiny dojde k jejich vzájemnému vyloučení, jelikož sledovaný program bude nejdříve spuštěn s jedním a poté s druhým parametrem ze zadané skupiny. Každý sledovaný program bude opakovaně spuštěn pro zadané frekvence nebo pro všechny frekvence v zadaném rozsahu. Po proběhnutí všech sledování může být výsledek rovnou zpracován do požadovaného výstupu.



Obrázek 4.1: Struktura modulů výsledné sady skriptů



Obrázek 4.2: Průběh výsledného algoritmu

Kapitola 5

Experimentální ověření přínosu práce

5.1 Návrh testovacích případů

Prvním testovacím případem je benchmark Linpack představený v roce 1979 [16], jelikož se jedná o benchmark závislý na rychlosti procesoru. Linpack je benchmark založený na řešení lineárních rovnic a původně byl implemetován v jazyce Fortran. V našem případě použijeme jeho shared memory verzi od společnosti Intel [20].

Jako druhý případ byl zvolen program Stream [21], který měří propustnost paměti RAM a tedy je ovlivněn její rychlostí více než rychlostí procesoru.

Prvním ze zástupců jednovláknových případů je benchmark Whetstone [26], který slouží k výpočtu MIPS (miliony instrukcí za sekundu) a jeho zdrojový kód byl pozměněn tak, že byl přidán kód pro inicializaci a provedení měření pomocí knihovny PAPI.

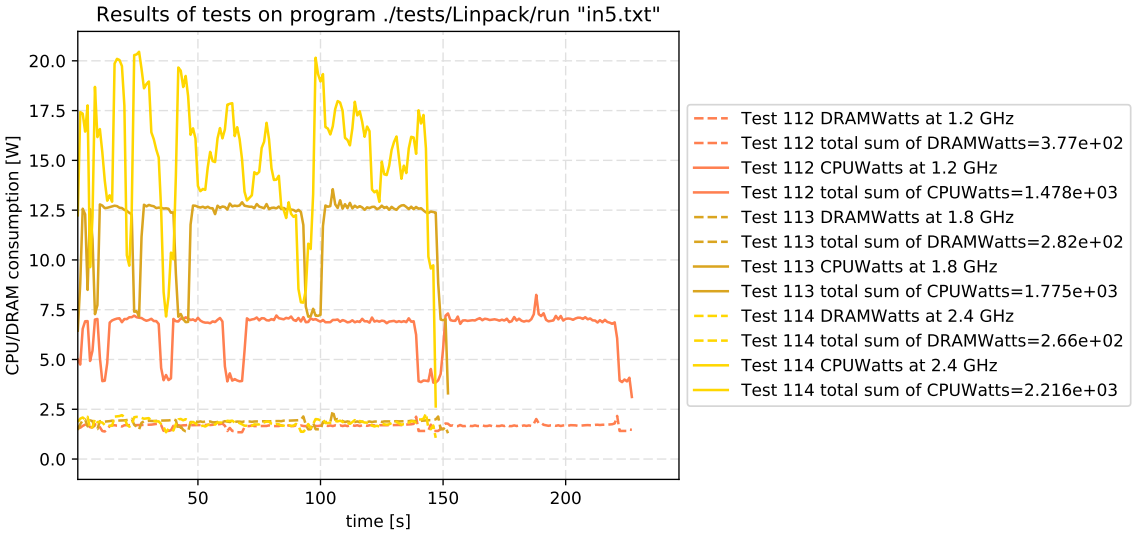
Druhý jednovláknový případ je jednoduché násobení matic maticemi. Cílem případu je využití paměti cache a byl proto opatřen kódem pro využití knihovny PAPI. Jelikož se jedná o program vytvořený přesně pro potřeby tohoto případu tak je možné specifikovat velikost matic, čímž se určí zda se matice vejde do paměti cache, a počet opakování, které je nutné pro efektivní využití paměti cache.

5.2 Testovací počítače

K testování již uvedených případů byl primárně využit notebook s procesorem z generace Haswell [18] a školní server vybavený také procesorem generace Haswell. Kompletní informace o počítačích využitých k testování naleznete v tabulce 5.1.

Tabulka 5.1: Softwarové a hardwarové vybavení testovacích počítačů

	Notebook	**Server**
Operační systém	Linux Mint 18	Ubuntu 16.04.4 LTS
Jádro OS	4.4.0-51-generic	4.4.0-122-generic
Procesor	Intel core i5-4200m	2xIntel Xeon E5-2620v3
RAM	8GB	2x32GB
PAPI	5.5.0	5.5.1
Python	3.5.2	3.5.2



H

Obrázek 5.1: Benchamrk Linpack na frekvencích 1.2 GHz, 1.8 GHz a 2.4 GHz jakožto zástupce benchmarků zatěžujících především procesor.

5.3 Výsledky testů

V této části budou provedeny experimentální měření testovacích případů z kapitoly 5.1 pomocí implementované sady skriptů a její výstupy zde budou předvedeny s podrobným vysvětlením vlivu jednotlivých případů na procesor a paměť.

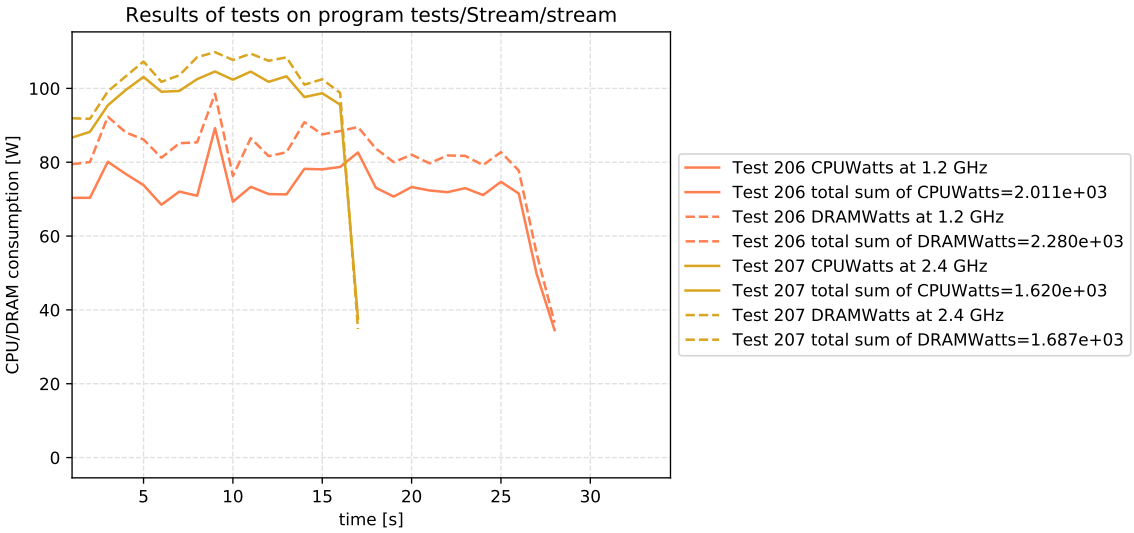
Jako první test pro ukázku funkcionality výsledné sady skriptů byl zvolen benchmark Linpack. Na grafu 5.1 lze jasně vidět, že benchmark je více vázán na rychlost procesoru než na paměť. V případě porovnání všech tří frekvencí je očividné, že zrychlením frekvence procesoru dojde ke zkrácení doby potřebné pro výpočet a neúměrně stoupne spotřeba procesoru.

Druhým testem, také určeným na ukázku výstupních grafů z výsledné sady skriptů, se stal benchmark Stream. Graf 5.2 ukazuje značně zvýšenou spotřebu paměti RAM oproti grafu 5.1, kde vyšší spotřebu má procesor.

Benchmark Whetstone představuje program, který zatěžuje pouze jedno jádro procesoru a lze vidět rozdíl spotřeby procesoru pokud je použito jen jedno jádro, jako v grafu 5.3, a nebo celý procesor, což lze vidět v grafu 5.1. Zároveň na něm lze ukázat typický výstup knihovny PAPI, představené v kapitole 3.1, který do popisu grafu přidá nové položky událostí, které byly na testovaném programu sledovány. V grafech 5.4 a 5.5 je tento přidaný výstup vidět a lze jednoduše porovnat hodnoty získané těmito sledováními. Graf 5.4 ukazuje množství správně odhadnutých a špatně odhadnutých predikcí podmíněných skoků, zatímco graf 5.5 obsahuje počet provedených instrukcí a reálný počet taktů procesoru.

Druhým hlavním výstupem z výsledné sady skriptů je CSV tabulka 5.2, která v tomto případě obsahuje informace o testu, ze kterého vznikla část grafu 5.3.

U experimentu s testovacím programem založeným na násobení matic budeme sledovat počet provedených instrukcí a skutečný počet taktů procesoru při zachování stejných argumentů a stejné frekvence. Jak můžeme vidět na grafu 5.6, počet instrukcí a počet taktů



Obrázek 5.2: Benchmark Stream na frekvencích 1GHz a 2GHz jakožto zástupce benchmarků zatěžujících především paměť. Tento test byl naměřen na serveru z důvodu vyšší spotřeby paměťových modulů.

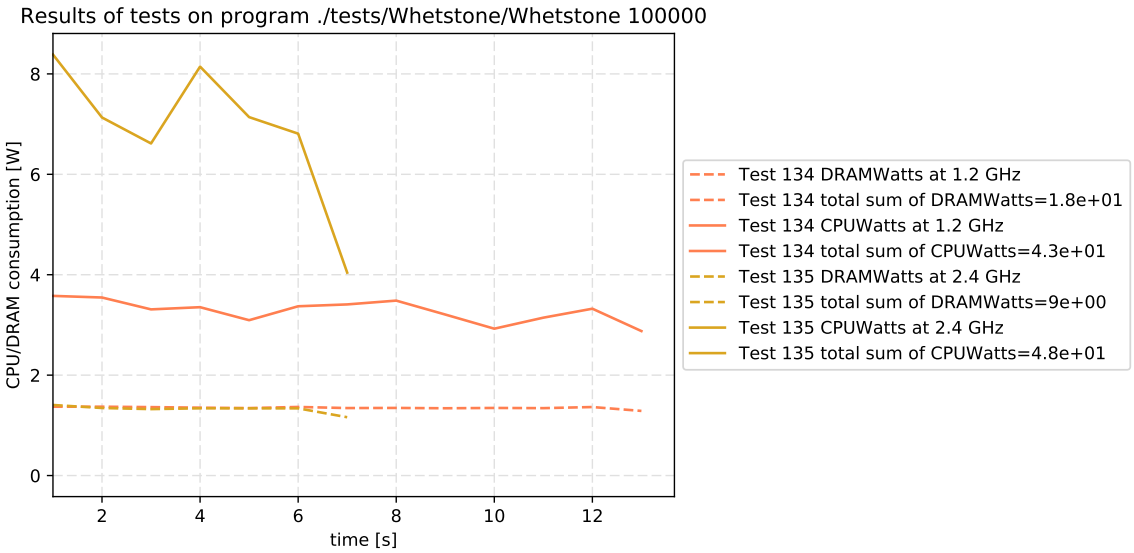
Tabulka 5.2: Benchmark Whetstone na frekvenci 2.4 GHz.

CPUWatts	**DRAMWatts**	**Temperature**
8.38562	1.405518	61.0
7.131897	1.345337	60.0
6.614502	1.323425	59.0
8.145264	1.339539	64.0
7.139343	1.338501	60.0
6.809875	1.338501	59.0
4.042664	1.161743	55.0

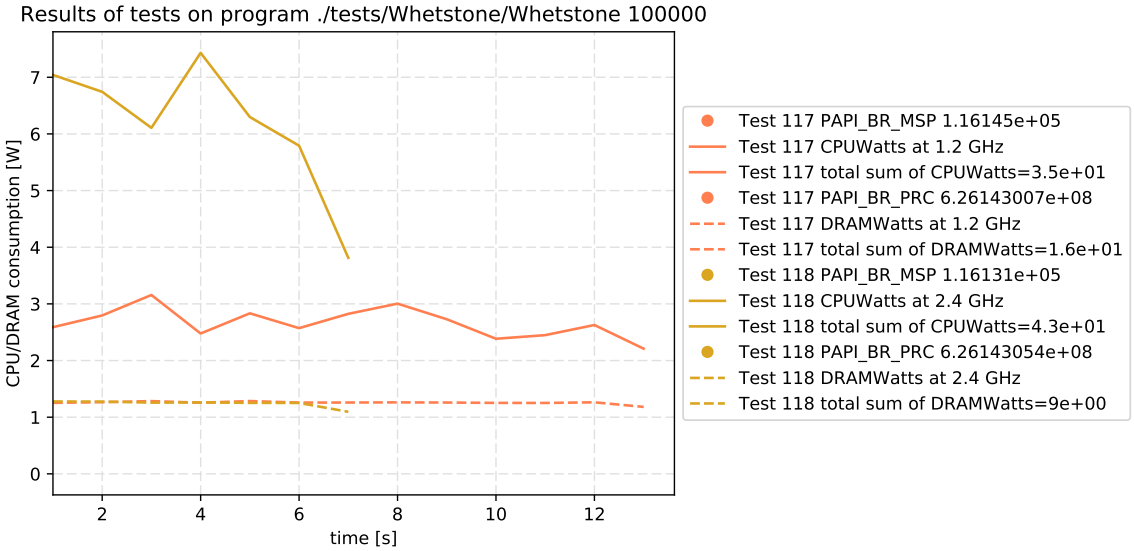
se liší jen velmi málo, což ukazuje velmi dobrou predikci moderních procesorů, jelikož bylo vykonáno pouze velmi málo instrukcí, u kterých byl zahozen jejich výsledek.

Sada skriptů také může generovat graf založený na stejné frekvenci a jeho obsahem budou všechny programy, které byly spuštěny ke sledování. Tento výstup můžeme vidět na grafu 5.7.

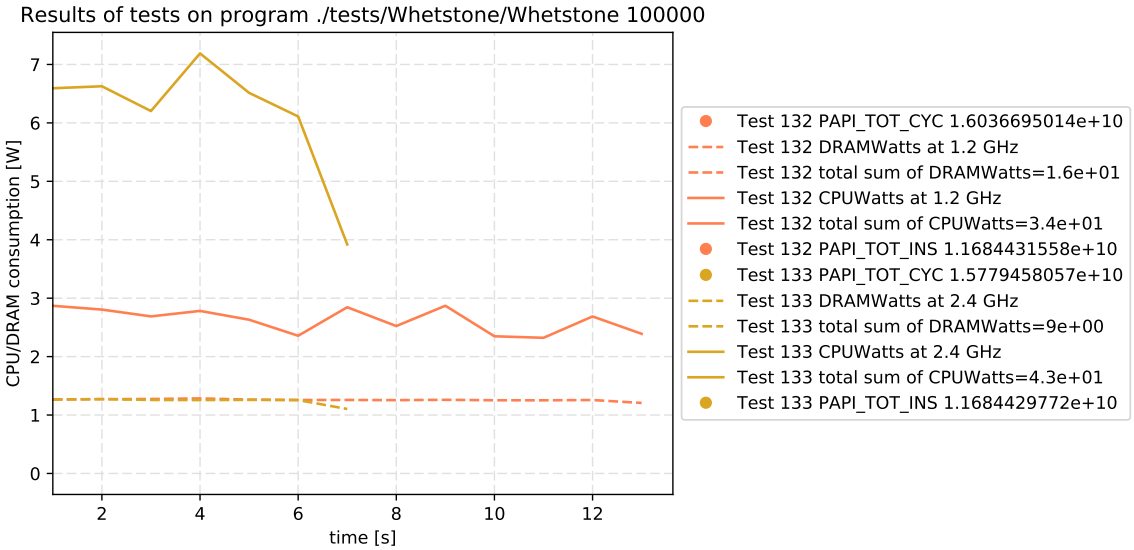
V posledním experimentu, který ukazuje funkcionalitu výsledné sady skriptů, se podíváme na grafy 5.5 a 5.6, na kterých lze vidět, že `PAPI_TOT_CYC` se na stejné frekvenci mění jen velmi málo, ale zároveň při změně frekvence zůstává podobný, což není ovlivněno sledovací sadou skriptů, ale implementací čítače na daném procesoru.



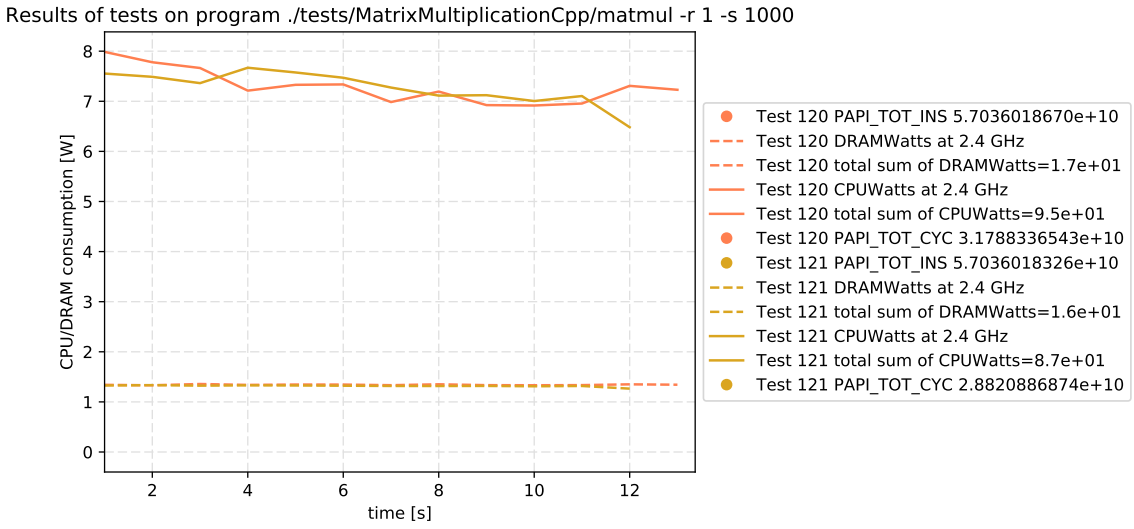
Obrázek 5.3: Whetstone na frekvencích 1.2 GHz a 2.4 GHz jakožto zástupce benchmarků zatěžujících pouze jedno vlákno (jádro) procesoru.



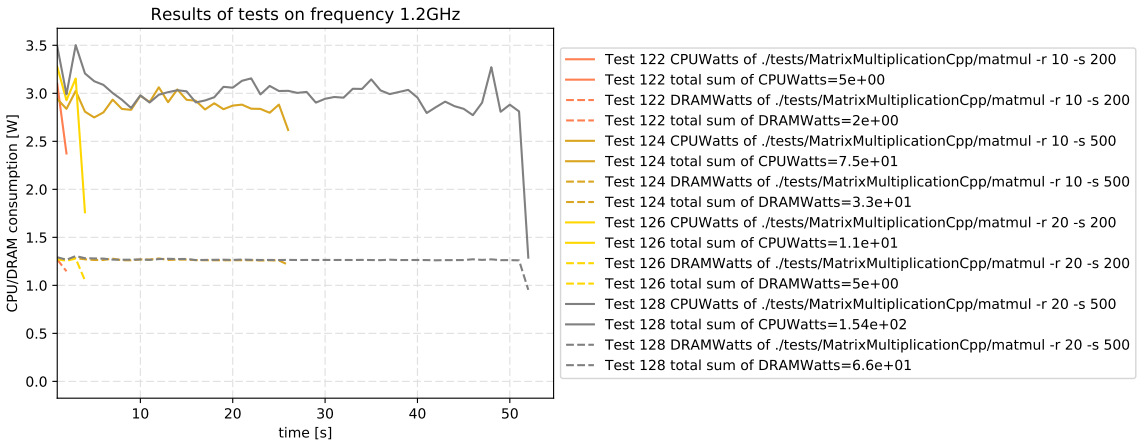
Obrázek 5.4: Whetstone stejně jako v předchozím případě, tentokrát spuštěný se sledováním PAPI událostí týkajících se predikce podmíněných skoků.



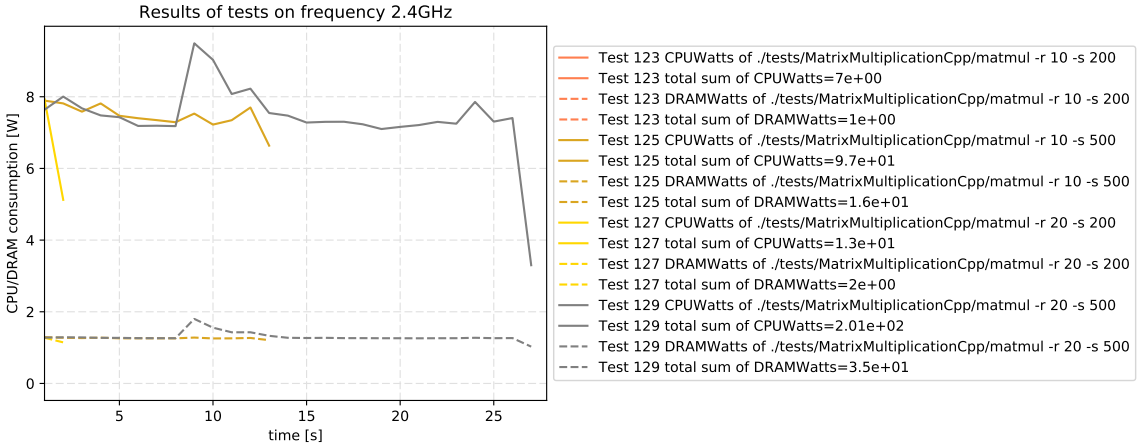
Obrázek 5.5: Whetstone stejně jako v předchozím případě, tentokrát spuštěný se sledováním PAPI událostí týkajících se počtu provedených instrukcí a proběhlých taktů procesoru.



Obrázek 5.6: Test založený na násobení matic maticemi spuštěný dvakrát po sobě se stejnými parametry a na stejné frekvenci. Povšimněte si velmi podobného počtu vykonaných instrukcí a uběhnutých cyklů procesoru.



(a) 1200 MHz



(b) 2400 MHz

Obrázek 5.7: Výstup programu seskupující programy spuštěné na stejné frekvenci.

Kapitola 6

Závěr

Cílem této práce bylo vytvoření univerzálního terminálového programu, který se bude skládat z několika skriptových modulů a předvedení naimplementovaných funkcí na známých bechmarcích a vyvození doporučení založených na výsledcích měření výsledným programem.

Hlavní cíl a to vytvoření sady skriptů určené ke sledovaní spotřeby běžícího programu splněn byl a tato sada je využitelná k měření spotřeby procesoru a paměti. V kapitole 5 byla práce s touto sadou skriptů předvedena včetně jejího výstupu pro uživatele a to v podání grafů a tabulek.

Kompletní implementace proběhla pouze pro procesory od společnosti Intel, jelikož zastupují většinu procesorů použitých v nejvýkonnějších superpočítačích světa, a pro operační systém Linux, který je také použit na většině z již zmíněných superpočítačů. Jádro skriptů je však multiplatformní a proto by bylo vhodné v budoucnu tuto práci rozšířit o podporu například procesorů ARM, které by mohly být v budoucnu použity jako alternativa k procesorům s instrukční sadou x86. Pro podporu architektury ARM prakticky chybí pouze měření spotřeby, jelikož vše ostatní je multiplatformní.

V experimentech byl ukázán výstup implementované sady skriptů a byly poukázány některé vlastnosti moderních procesorů jako například přesnost predikce výsledku podmínek u podmíněných skoků.

Literatura

[1] ABOUT THE MONT-BLANC PROJECTS. [Online; navštíveno 01.05.2018].
URL http://montblanc-project.eu/project/presentation#project-history

[2] CPU-World. [Online; navštíveno 08.05.2018].
URL www.cpu-world.com

[3] Efficient UPS Aids Google's Extreme PUE. [Online; navštíveno 08.05.2018].
URL http://www.datacenterknowledge.com/archives/2009/04/01/efficient-ups-aids-googles-extreme-pue

[4] Intel® Performance Counter Monitor - A better way to measure CPU utilization.
URL https://software.intel.com/en-us/articles/intel-performance-counter-monitor

[5] Performance Application Programming Interface Architecture. [Online; navštíveno 08.05.2018].
URL http://icl.cs.utk.edu/projects/papi/files/documentation/PAPI_USER_GUIDE_23.htm

[6] Performance Application Programming Interface Overview. [Online; navštíveno 08.05.2018].
URL http://icl.cs.utk.edu/projects/papi/wiki/PAPIC:Overview#Events

[7] SYSFS(5) Linux Programmer's Manual. [Online; navštíveno 08.05.2018].
URL http://man7.org/linux/man-pages/man5/sysfs.5.html

[8] Top500 List. [Online; navštíveno 08.05.2018].
URL https://www.top500.org/

[9] Top500 List - November 2017. [Online; navštíveno 08.05.2018].
URL https://www.top500.org/list/2017/11/

[10] Přednáška č. 1 předmětu INP. FIT VUT v Brně, 2017.

[11] Přednáška č. 12 předmětu INP. FIT VUT v Brně, 2017.

[12] ARM: Memory access instructions. [Online; navštíveno 08.05.2018].
URL https://developer.arm.com/products/architecture/a-profile/docs/100898/latest/memory-access-instructions

[13] Brinn, L. W.: Boolean algebra. *International Journal of Mathematical Education in Science and Technology*, ročník 20, č. 6, 1989: s. 799–807,

doi:10.1080/0020739890200602, https://doi.org/10.1080/0020739890200602.
URL https://doi.org/10.1080/0020739890200602

[14] Chen, J.; Gupta, A. K.: Testing and Locating Variance Changepoints with Application to Stock Prices. *Journal of the American Statistical Association*, ročník 92, č. 438, 1997: s. 739–747, doi:10.1080/01621459.1997.10474026, https://doi.org/10.1080/01621459.1997.10474026.
URL https://doi.org/10.1080/01621459.1997.10474026

[15] Cleveland, C. J.; Morris, C.: Section 10 - Electricity. In *Handbook of Energy*, editace C. J. Cleveland; C. Morris, Boston: Elsevier, 2014, ISBN 978-0-12-417013-1, s. 169 – 195, doi:https://doi.org/10.1016/B978-0-12-417013-1.00010-8.
URL
https://www.sciencedirect.com/science/article/pii/B9780124170131000108

[16] Dongarra JJ, M. C. S. G., Bunch J: *LINPACK User's Guide*. SIAM: Philadelphia, PA, 1979, ISBN 9781611971811.

[17] Fu, L. J. Y. J. e. a., H.: The Sunway TaihuLight supercomputer: system and applications. *China Inf. Sci. 59: 072001*, 2016, [Online; navštíveno 01.05.2018].
URL https://doi.org/10.1007/s11432-016-5588-7

[18] Hammarlund, P.; Martinez, A. J.; Bajwa, A. A.; aj.: Haswell: The Fourth-Generation Intel Core Processor. *IEEE Micro*, ročník 34, č. 2, Mar 2014: s. 6–20, ISSN 0272-1732, doi:10.1109/MM.2014.10.

[19] Intel: Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B part 2. [Online; navštíveno 08.05.2018].
URL
https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

[20] Intel®: Intel® Math Kernel Library Benchmarks (Intel® MKL Benchmarks). [Online; navštíveno 01.05.2018].
URL
https://software.intel.com/en-us/articles/intel-mkl-benchmarks-suite

[21] John D. McCalpin, P.: STREAM: Sustainable Memory Bandwidth in High Performance Computers. [Online; navštíveno 01.05.2018].
URL https://www.cs.virginia.edu/stream/

[22] Jovanović, L. D.: A Novel TDM-Based High-Precision Wattmeter. *IEEE Transactions on Instrumentation and Measurement*, ročník 66, č. 6, June 2017: s. 1083–1088, ISSN 0018-9456, doi:10.1109/TIM.2017.2653458.

[23] Kemp, A. J.; Valentine, G. J.; Hopkins, J. M.; aj.: Thermal management in vertical-external-cavity surface-emitting lasers: finite-element analysis of a heatspreader approach. *IEEE Journal of Quantum Electronics*, ročník 41, č. 2, Feb 2005: s. 148–155, ISSN 0018-9197, doi:10.1109/JQE.2004.839706.

[24] Kuhn, T.-J. K. L. K.: *CMOS and beyond : logic switches for terascale integrated circuits*. Cambridge University Press, 2015, ISBN 9781107043183.

[25] Masood, F.: RISC and CISC. *ArXiv e-prints*, Leden 2011, 1101.5364.

[26] Painter Engineering, I.: Whetstone benchmark. [Online; navštíveno 01.05.2018].
URL http://www.netlib.org/benchmark/whetstone.c

[27] Shimokawabe, T.; Aoki, T.; Ishida, J.; aj.: 145 TFlops Performance on 3990 GPUs of TSUBAME 2.0 Supercomputer for an Operational Weather Prediction. *Procedia Computer Science*, ročník 4, 2011: s. 1535 – 1544, ISSN 1877-0509, doi:https://doi.org/10.1016/j.procs.2011.04.166, proceedings of the International Conference on Computational Science, ICCS 2011.
URL http://www.sciencedirect.com/science/article/pii/S1877050911002249

[28] Chapter 3 - Intel® Core™ Processors. In *Power and Performance in Enterprise Systems*, editace J. Kukunas, Boston: Morgan Kaufmann, 2015, ISBN 978-0-12-800726-6, s. 43 – 52, doi:https://doi.org/10.1016/B978-0-12-800726-6.00003-3.
URL
https://www.sciencedirect.com/science/article/pii/B9780128007266000033

[29] Terpstra, D.; Jagode, H.; You, H.; aj.: Collecting Performance Data with PAPI-C. In *Tools for High Performance Computing 2009*, editace M. S. Müller; M. M. Resch; A. Schulz; W. E. Nagel, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ISBN 978-3-642-11261-4, s. 157–173.

[30] Thompson, M. T.: Chapter 4 - Bipolar Transistor Models. In *Intuitive Analog Circuit Design*, editace M. T. Thompson, Boston: Newnes, 2014, ISBN 978-0-12-405866-8, s. 87 – 117, doi:https://doi.org/10.1016/B978-0-12-405866-8.00004-8.
URL
https://www.sciencedirect.com/science/article/pii/B9780124058668000048

[31] TOP500: Sunway TaihuLight. [Online; navštíveno 01.05.2018].
URL https://www.top500.org/system/178764

[32] Wahlroos, M.; Pärssinen, M.; Manner, J.; aj.: Utilizing data center waste heat in district heating – Impacts on energy efficiency and prospects for low-temperature district heating networks. *Energy*, ročník 140, 2017: s. 1228 – 1238, ISSN 0360-5442, doi:https://doi.org/10.1016/j.energy.2017.08.078.
URL http://www.sciencedirect.com/science/article/pii/S0360544217314548

[33] ČEZ: Nejen Ohmův zákon - fyzikální základy. [Online; navštíveno 01.05.2018].
URL https://www.cez.cz/edee/content/microsites/elektrina/fyz3.htm

Příloha A

Manuál k použití

Před prvním spuštěním sady skriptů je zapotřebí nainstalovat vyžadované závislosti. Těmito závislostmi jsou Python 3.5, Tkinter, Matplotlib a Numpy. Pro jednoduché nainstalování lze na CD nalézt soubor `instructions.txt`, ve kterém jsou již vytvořeny příkazy pro OS Debian nebo Ubuntu a jejich deriváty.

Ke spuštění je vhodné použít skript `run`, který načte požadované moduly kernelu a vyžádá si od uživatele oprávnění na úroveň uživatele `root` pomocí příkazu `sudo`. Tento skript poté spustí hlavní soubor umístěný v adresáři `src`.

Základní způsoby použití lze nalézt v souboru `README.md`. Pro kompletní výpis všech přepínačů skriptu spustíme příkaz A.1 nebo A.2 pro výpis nápovědy pro daný modul.

```
1 ./run --help
```

Výpis A.1: Výpis nápovědy

```
1 ./run modul --help
```

Výpis A.2: Výpis nápovědy pro vybraný modul